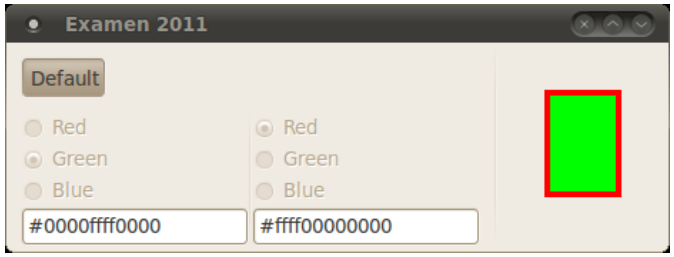
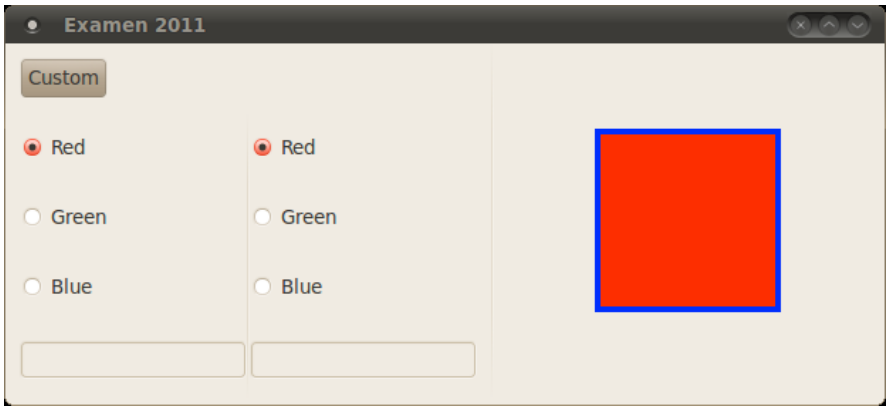
Gtk::Examen

Tout document autorisé, barème indicatif.

Dans cet examen, on vous invite à concevoir l'interface graphique illustrée ci-dessous. L'interface est représentée avec deux tailles de fenêtre :





Cette interface visualise un carré dont les couleurs de remplissage et de bord sont définies par le « formulaire » de la partie gauche de l'interface. Le bouton « Custom/Default » est un bouton à deux états (`Gtk::ToggleButton`) lorsqu'il est enfoncé (*cf.* Figure du haut), il permet à l'utilisateur de donner les couleurs de remplissage et de bord en utilisant des codes hexadécimaux dans les `Gtk::Entry`. Lorsque le bouton n'est pas enfoncé (*cf.* Figure du bas), l'utilisateur choisit ses couleurs parmi les couleurs par défaut.

Quelques précisions sur le comportement de l'interface :

- Lorsque l'utilisateur utilise le bouton « Custom/Default », on passe alternativement entre une configuration des couleurs par défaut ou manuelle.
- À chaque modification d'une couleur (changement de choix d'une couleur par défaut ou modification du texte des `Gtk::Entry`) alors le dessin est automatiquement mis à jour.
- Lorsque l'utilisateur utilise une couleur par défaut, celle-ci est reportée automatiquement dans le `Gtk::Entry` correspondant si le code couleur n'ai pas acceptable, alors la couleur utilisée est noir.
- Le carré est dessiné avec les coordonnées des coins suivantes : *(width/4, height/4), (3*width/4, 3*height/4)*, où *width* et *height* sont les caractéristiques géométriques de la zone de dessin.
- À propos de la géométrie (ce qu'il faut remarquer des deux figures) :
  - Les lignes verticales sont ajoutées en utilisant des widgets de classe

`Gtk::VSeparator` et **peuvent être omises dans vos réponses** mais elles désambigüisent certains choix.

- Le bouton « Custom/Default » ne bouge pas et ne change pas de taille (verticale comme horizontale)
- Les `Gtk::Entry` ne change pas de taille (verticale comme horizontale)
- Seule la largeur de la zone de dessin est modifiée

*Q.1) (**3 points**) Construire un arbre représentant la structure hiérarchique des widgets de l'interface. Vous indiquerez les types de* packing *qui permettront d'obtenir le comportement géométrique de l'interface définis plus haut.*

*Q.2) (**4 points**) En déduire la fonction* `constructWindow()` *qui réalise l'interface telle qu'illustrée ci-dessus (vous pouvez vous limiter à décrire l'organisation des RadioButton que pour une couleur).*

*Q.3) (**2 points**) Compléter la classe* `DessinCarre` *pour que le dessin d'un carré soit réalisé dans la zone de visualisation de l'interface.*

*Q.4) (**3 points**) Écrire la fonction* `declareConnections()` *dans laquelle vous écrirez les connexions qui permettent à l'interface de réagir tel que cela a été décris ci-dessus.*

On vous fournit également les éléments suivants :

- une version à compléter du code pour les classes `DessinCarre` et `fenetreExam2011` : **pour éviter de recopier tout le code, vous indiquerez sur votre copie le code à insérer pour chaque position repérée dans les sources par un commentaire**. Les points d'insertions peuvent servir à plusieurs questions.
- la fonction `Gtk::Widget::set_sensitive(bool)` permet de définir si un widget est sensible ou non, c'est-à-dire actif pour l'interaction ou non.
- un extrait de la documentation à propos de la classe `Gtk::ToggleButton`.

Documentation pour la classe `Gtk::Entry` :

Les signaux disponibles pour ma classe `Gtk::Entry` sont les suivants :

- `Glib::SignalProxy0<void> signal_activate()`

Le signal `activate` désigne des envois d'évènements à chaque modification de texte.

- `Glib::SignalProxy1<void,const Glib::ustring&> signal_insert_at_cursor()`

Le signal `insert_at_cursor` déclenche des évènements lors de l'insertion d'un texte (donné en paramètre).

Documentation pour la classe `Gtk::ToggleButton` :

- `Glib::SignalProxy0<void> signal_toggled()`

Le signal `toggled` déclenche des évènements lors de la modification de l'état d'un `ToggleButton`.

```
#include "gtkmm.h"

class DessinCarre : public Gtk::DrawingArea
{
protected:
        Gdk::Color background, foreground;
public:
        DessinCarre():Gtk::DrawingArea(){
                background.set_rgb(65000,12000,0);
                foreground.set_rgb(0,12000,65000);
        };
        virtual ~DessinCarre(){};

        Gdk::Color getBG() {return background;};
        Gdk::Color getFG() {return foreground;};

        void setBG(Gdk::Color c) {
                background = c;
                queue_draw();
        };
        void setFG(Gdk::Color c) {
                foreground = c;
                queue_draw();
        };

        // POINT d'INSERTION DessinCarre_decl //
};

class fenetreExam2011 : public Gtk::Window
{
public:
        fenetreExam2011();
        virtual ~fenetreExam2011(){};

protected:
        void constructWindow();
        void declareConnections();

        void doToggle();
        void setCustomColor();
        void setCustomColorText();

protected:
        Gtk::RadioButton *button_BG_Red, *button_BG_Green, *button_BG_Blue;
        Gtk::RadioButton *button_FG_Red, *button_FG_Green, *button_FG_Blue;
        DessinCarre darea;
        Gtk::ToggleButton togglebutton;
        Gtk::Entry entry_BG, entry_FG;

        // POINT d'INSERTION FenetreExam_decl //
};

void fenetreExam2011::constructWindow()
{
        // POINT d'INSERTION ConstructWindow //
}

void fenetreExam2011::declareConnections()
{
        // POINT d'INSERTION DeclareConnexion //
}

void fenetreExam2011::doToggle()
{
        if( togglebutton.get_active() ) {
                togglebutton.set_label("Default");
                button_BG_Red->set_sensitive(false);
                button_BG_Green->set_sensitive(false);
                button_BG_Blue->set_sensitive(false);
                button_FG_Red->set_sensitive(false);
                button_FG_Green->set_sensitive(false);
                button_FG_Blue->set_sensitive(false);
                entry_BG.set_sensitive();
                entry_FG.set_sensitive();
```

```
75          } else {
76                  togglebutton.set_label("Custom");
77                  button_BG_Red->set_sensitive();
78                  button_BG_Green->set_sensitive();
79                  button_BG_Blue->set_sensitive();
80                  button_FG_Red->set_sensitive();
81                  button_FG_Green->set_sensitive();
82                  button_FG_Blue->set_sensitive();
83                  entry_BG.set_sensitive(false);
84                  entry_FG.set_sensitive(false);
85          }
86  }
87
88  void fenetreExam2011::setCustomColorText()
89  {
90          entry_BG.set_text( darea.getBG().to_string() );
91          entry_FG.set_text( darea.getFG().to_string() );
92  }
93
94  void fenetreExam2011::setCustomColor()
95  {
96          darea.setBG( Gdk::Color(entry_BG.get_text()) );
97          darea.setFG( Gdk::Color(entry_FG.get_text()) );
98  }
99
100 fenetreExam2011::fenetreExam2011() : Gtk::Window()
101 {
102         constructWindow();
103         declareConnections();
104 }
105
106
107 int main(int argc, char** argv)
108 {
109         Gtk::Main kit(argc, argv);
110         fenetreExam2011 fenetre;
111         Gtk::Main::run(fenetre);
112         return 0;
113 }
```